**Text Summarization**

**Abstract:**

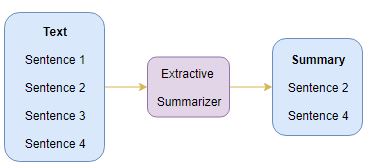
In this new era, where tremendous information is available on the internet, it is most important to provide the improved mechanism to extract the information quickly and most efficiently. It is very difficult for human beings to manually extract the summary of large documents of text. There are plenty of text material available on the internet. So there is a problem of searching for relevant documents from the number of documents available, and absorbing relevant information from it.In order to solve the above two problems, the automatic text summarization is very much necessary. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings.

**Introduction:**

Before going to the Text summarization, first we, have to know that what a summary is. A summary is a text that is produced from one or more texts, that conveys important information in the original text, and it is of a shorter form. The goal of automatic text summarization is presenting the source text into a shorter version with semantics. The most important advantage of using a summary is, it reduces the reading time. Text Summarization methods can be classified into extractive and abstractive summarization. An extractive summarization method consists of selecting important sentences, paragraphs etc. from the original document and concatenating them into shorter form. An Abstractive summarization is an understanding of the main concepts in a document and then express those concepts in clear natural language. There are two different groups of text summarization: indicative and informative. Inductive summarization only represent the main idea of the text to the user. The typical length of this type of summarization is 5 to 10 percent of the main text. On the other hand, the informative summarization systems gives concise information of the main text .The length of informative summary is 20 to 30 percent of the main text.

There are broadly two different approaches that are used for text summarization:

* Extractive Summarization
* Abstractive Summarization

**Extractive Summarization:** The name gives away what this approach does. We identify the important sentences or phrases from the original text and extract only those from the text. Those extracted sentences would be our summary. The below diagram illustrates extractive summarization:

### abstractive summarizationAbstractive Summarization: This is a very interesting approach. Here, we generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were present. The sentences generated through abstractive summarization might not be present in the original text:

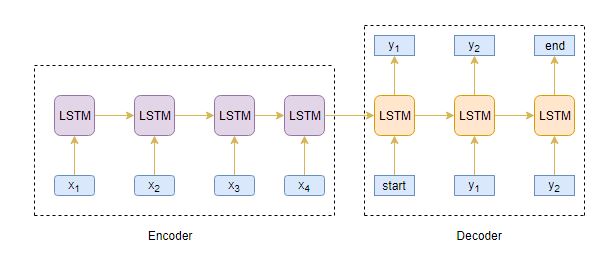
## **Introduction to Sequence-to-Sequence (Seq2Seq) Modeling**

We can build a Seq2Seq model on any problem, which involves sequential information. This includes Sentiment classification, Neural Machine Translation, and Named Entity Recognition – some very common applications of sequential information.

nmtIn the case of Neural Machine Translation, the input is a text in one language and the output is also a text in another language:

nerIn the Named Entity Recognition, the input is a sequence of words and the output is a sequence of tags for every word in the input sequence:

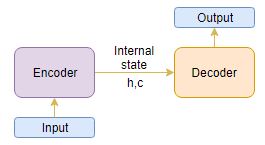
Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, **we can model this as a Many-to-Many Seq2Seq problem.** Below is a typical Seq2Seq model architecture:



There are two major components of a Seq2Seq model:

* Encoder
* Decoder

The Encoder-Decoder architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences are of different lengths.

Let us understand this from the perspective of text summarization. The input is a long sequence of words and the output will be a short version of the input sequence.

Generally, variants of Recurrent Neural Networks (RNNs), i.e. Gated Recurrent Neural Network (GRU) or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long-term dependencies by overcoming the problem of vanishing gradient.

We can set up the Encoder-Decoder in 2 phases:

* Training phase
* Inference phase

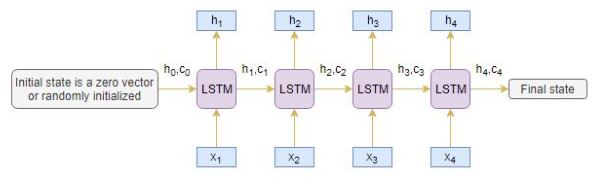
### Training phase

In the training phase, we will first set up the encoder and decoder. We will then train the model to predict the target sequence offset by one timestamp. Let us see in detail on how to set up the encoder and decoder.

**Encoder**

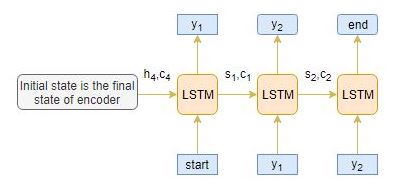
An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestamp, one word is fed into the encoder. It then processes the information at every timestamp and captures the contextual information present in the input sequence.

I have put together the below diagram which illustrates this process:



The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder. Remember, this is because the encoder and decoder are two different sets of the LSTM architecture.

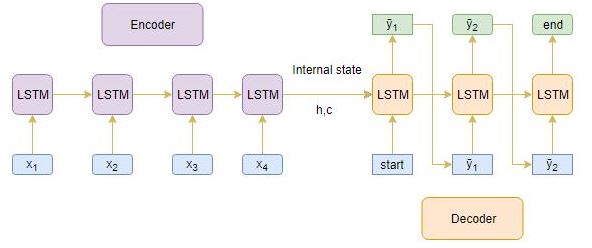
**Decoder**

The decoder is also an LSTM network, which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestamp. **The decoder is trained to predict the next word in the sequence given the previous word.**

**<start>** and <**end>**are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <**start>**token. And the <**end>**token signals the end of the sentence.

### Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence:



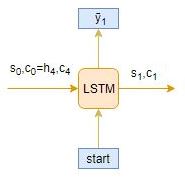
**How does the inference process work?**

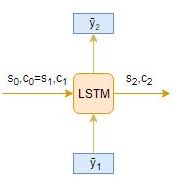
Here are the steps to decode the test sequence:

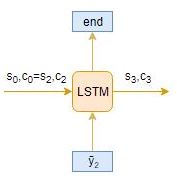
1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Pass <**start>** token as an input to the decoder
3. Run the decoder for one timestep with the internal states
4. The output will be the probability for the next word. The word with the maximum probability will be selected
5. Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step
6. Repeat steps 3 – 5 until we generate <**end>** token or hit the maximum length of the target sequence

Let’s take an example where the test sequence is given by  [x1, x2, x3, x4]. How will the inference process work for this test sequence? I want you to think about it before you look at my thoughts below.

1. Encode the test sequence into internal state vectors
2. Observe how the decoder predicts the target sequence at each timestep:

**Timestep: t=1**

**Timestep: t=2**

**Timestep: t=3**

## Limitations of the Encoder – Decoder Architecture

As useful as this encoder-decoder architecture is, there are certain limitations that come with it.

* The encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence. **This works only for short sequences** since the decoder is looking at the entire input sequence for the prediction
* Here comes the problem with long sequences. **It is difficult for the encoder to memorize long sequences into a fixed length vector**